28 August 2023 20:48

Functions

Function Overloading

OOPs → Class
Object
Class Methods

Arr: {1 2 3 4} Sum = 4

1 + 2 = 3
1 + 3 = 4 ✓
1 4

1, 3 Ans.

Arr: {1 2 3 4} Sum = 7

✓ 1+2 = 3 X
✓ 1+3 = 4 X
1+4 = 5 X

2+3 = 5 X
2+4 = 6 X

3+4 = 7
Ans.

Arr: {2 3 1 7}$_N$ Sum = 4 (i)

2 + 3 = 5 X
2 + 1 = 3 X
2 + 7 = 9 X

3+1 = 4 ✓

3, 1

first no. ← i → 0 to N-1

first no. ← i → 0 to N-1

Second no: i+1 to N-1

Functions → User defined functions

↓

Predefined functions

Reusability

max (a, b)

return type name (parameters) {

≡ Body

}

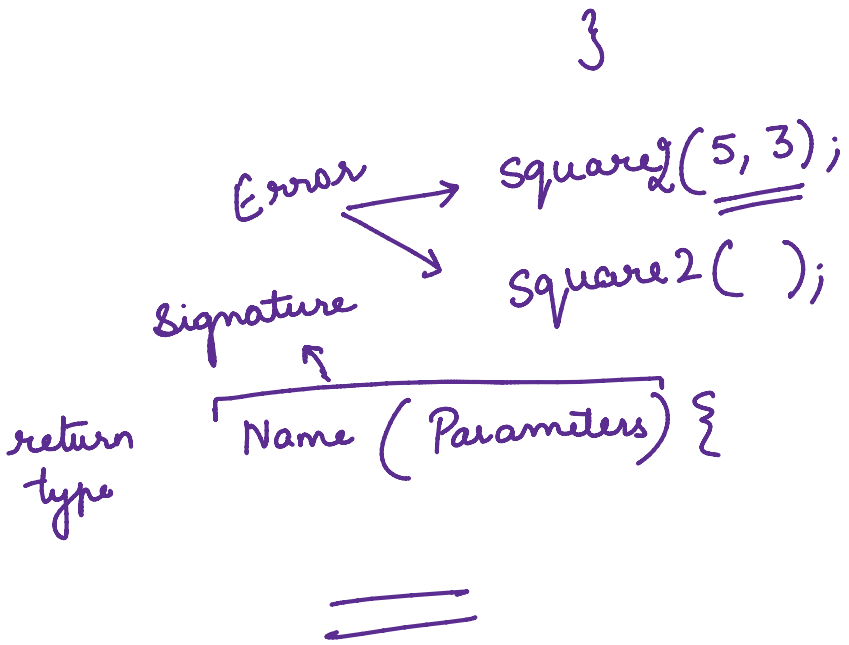
```
void square (int n) {
    cout << n*n;
}
```

square(3); → 9

square(7); → 49

```
int square2 (int n) {
    return n*n;
}
```

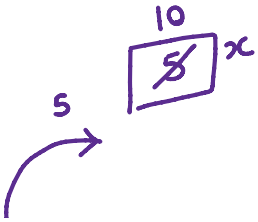
```
main() {
    cout << square2(4);
    int x = square2(5);
}
```




```
#include<iostream> //Header File
using namespace std;
int sum(int a,int b){
   return a+b;
}
int main(){ //Start of the main function
   cout<<sum(2,3);
 //  cout<<sum(1,2,3); //too many arguments to function 'int sum(int, int)'
  // cout<<sum(5); //too few arguments to function 'int sum(int, int)'
}
```

```
#include<iostream> //Header File
using namespace std;
int sum(int a,int b); //Declaration
int main(){ //Start of the main function
   cout<<sum(2,3);
}
int sum(int a,int b){ //Definition
   return a+b;
}
```

```
#include<iostream> //Header File
using namespace std;
int sum(int a,int b=5){ //Default Argument
   return a+b;
}
int main(){ //Start of the main function
   cout<<sum(2,3)<<endl;
   cout<<sum(7);
}
```

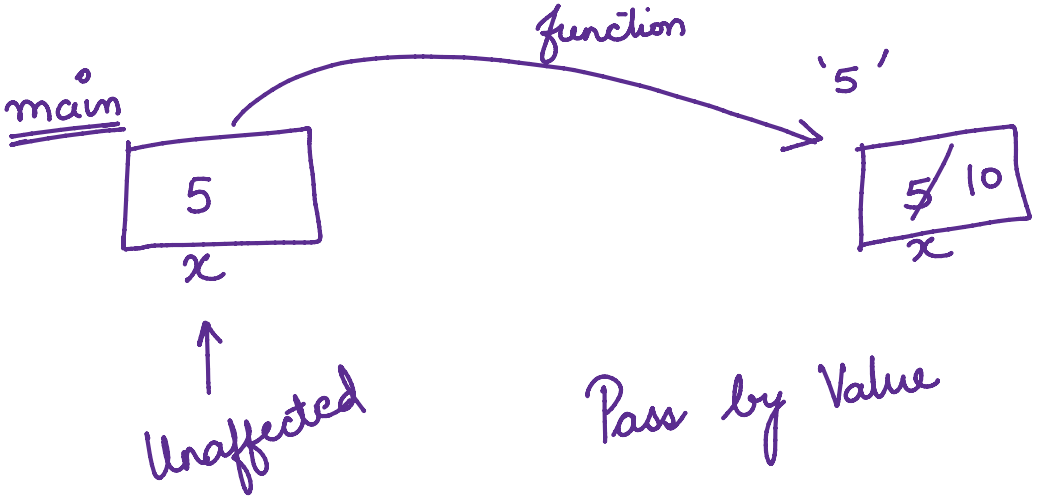



```
#include<iostream> //Header File
using namespace std;
void fun(int x){
   x=10;
   cout<<x<<endl;
}
int main(){ //Start of the main function
```

o/p: 5
10


```
using namespace std;
void fun(int x){
   x=10;
   cout<<x<<endl;
}
int main(){ //Start of the main function
   int x=5;
   cout<<x<<endl;
   fun(x);
   cout<<x<<endl;
}
```

o/p: 5
10
5

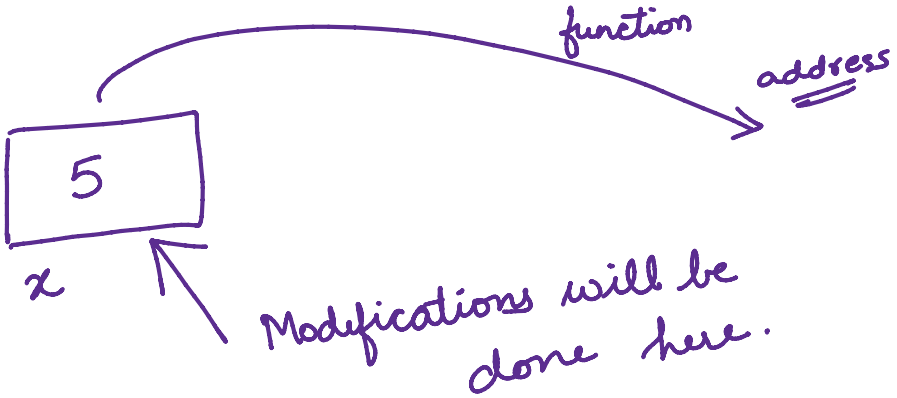


Unaffected

Pass by Value

Pass by reference

Address



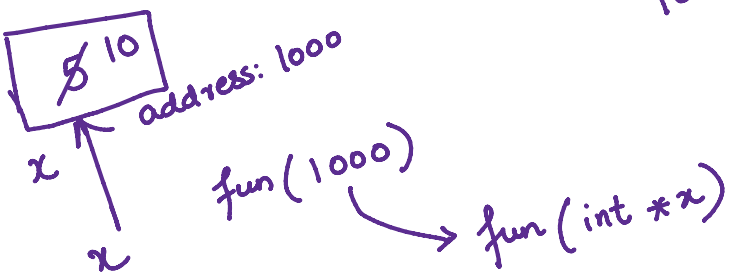
Modifications will be done here.

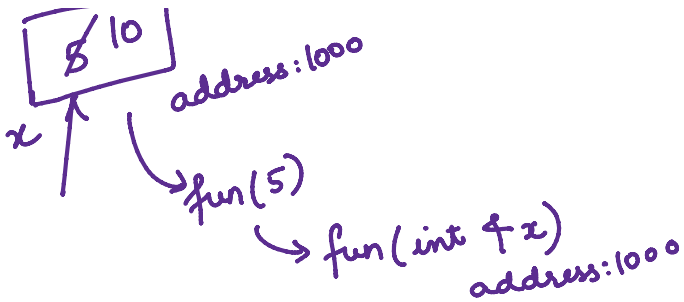
Pass By Reference

```
#include<iostream> //Header File
using namespace std;
void fun(int *x){
   *x=10;
   cout<<*x<<endl;
}
int main(){ //Start of the main function
   int x=5;
   cout<<x<<endl;
   fun(&x);
   cout<<x<<endl;
}
```

```
#include<iostream> //Header File
using namespace std;
void fun(int &x){
   x=10;
   cout<<x<<endl;
}
int main(){ //Start of the main function
   int x=5;
   cout<<x<<endl;
   fun(x);
   cout<<x<<endl;
}
```

o/p: 5
10
10





Sum of 2 numbers — Sum( )

Sum of 2 decimal values — Sum( )

Function Overloading

Function name is same

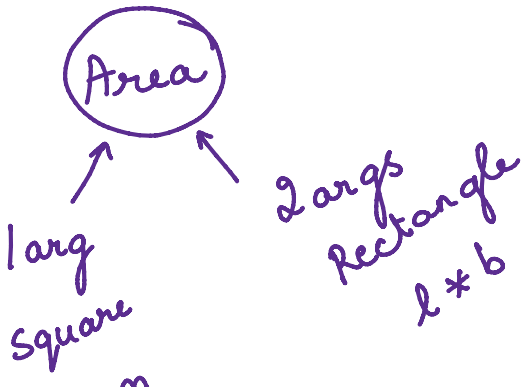
Different function signature

Name (Parameters)

Name: same

Parameters: Different

1.) Number of parameters
2.) Type of parameters
3.) Order of parameters



Square

$n * n$

```cpp
#include<iostream> //Header File
using namespace std;
void Area(int n){
   cout<<"First function: ";
   cout<<n*n;
}
void Area(int l,int b){
   cout<<"Second function: ";
   cout<<l*b;
}
int main(){ //Start of the main function
   Area(6);
   cout<<endl;
   Area(4,7);
}
```

First function

Second function

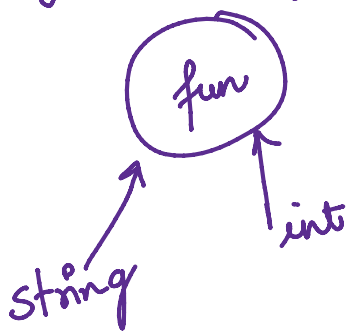
If `void Area(int l, int b=6){}`

$\equiv$

`}`

Area (6) → ERROR.

2. Type of Arguments

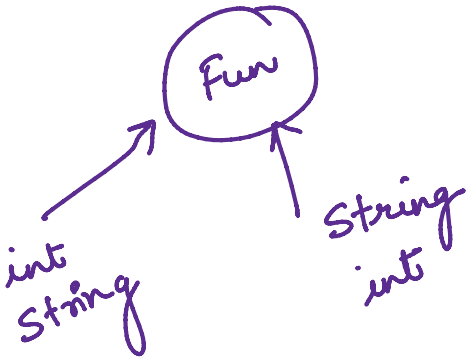



```cpp
#include<iostream> //Header File
using namespace std;
void Fun(int n){
   cout<<"First function: "<<endl;
}
void Fun(string s){
   cout<<"Second function: "<<endl;
}
int main(){ //Start of the main function
   Fun(6);
   Fun("Hello");
}
```

First function

Second functions

③ Order of arguments




```cpp
#include<iostream> //Header File
using namespace std;
void Fun(int n,string s){
   cout<<"First function: "<<endl;
}
void Fun(string s,int n){
   cout<<"Second function: "<<endl;
}
int main(){ //Start of the main function
   Fun(6,"ABC");
   Fun("Hello",8);
}
```

First fun

Second fun

```cpp
#include<iostream> //Header File
using namespace std;
void Fun(int n,string s){
   cout<<"First function: "<<endl;
}
int Fun(int n,string s){
   cout<<"Second function: "<<endl;
   return 0;
}
int main(){ //Start of the main function
   Fun(6,"ABC");
}
```

→ Error (not function overloading)

OOPs

Object Oriented Programming

Class → User defined data-type
Blueprint
(attributes, functions)

Object
↓
Instance of the class

```
class Student{
    int rollno;
    void print(){
        cout <<" Roll No" << rollno;
    }
};
```

```
Student s1, s2;
```


```
#include<iostream> //Header File
using namespace std;
class Student{
   public:
      int rollno; //Attribute
      void print(){ //Function
         cout<<"RollNo:"<<rollno<<endl;
      }
};
int main(){ //Start of the main function
   Student s1;
   s1.rollno=12;
   s1.print();
   Student s2;
   s2.rollno=15;
   s2.print();
}
```

Objects : classname obj ;

Attributes : obj. attribute ;

Functions : obj. function

"DOT"

Pointer to an object :

classname * ptr;

Attribute : ptr -> attribute

Functions : ptr -> function

ARROW ->

```
#include<iostream> //Header File
using namespace std;
class Student{
   public:
      int rollno; //Attribute
      void print(){ //Function
         cout<<"RollNo:"<<rollno<<endl;
      }
};
int main(){ //Start of the main function
   Student s1;
   s1.rollno=12;
   s1.print();
   Student s2;
   s2.rollno=15;
   s2.print();
   Student *s3;
   s3->rollno=10;
   s3->print();
}
```

"Constructors" : special type of function

1. Same name as that of Class.
2. Doesn't have a return type
3. Called automatically when object is created.

```
#include<iostream> //Header File
using namespace std;
class Student{
   public:
      int rollno; //Attribute
      void print(){ //Function
         cout<<"RollNo:"<<rollno<<endl;
      }
      Student(){
         cout<<"Constructor is called"<<endl;
      }
};
int main(){ //Start of the main function
   Student s1,s2;
}
```

```cpp
#include<iostream> //Header File
using namespace std;
class Student{
   public:
      int rollno; //Attribute
      void print(){ //Function
         cout<<"RollNo:"<<rollno<<endl;
      }
      Student(){
         cout<<"Default constructor is called"<<endl;
      }
      Student(int n){
         cout<<"Parameterized constructor is called"<<endl;
         rollno=n;
      }
};
int main(){ //Start of the main function
   Student s1;
   s1.print();
   Student s2(50);
   s2.print();
}
```

```cpp
#include<iostream> //Header File
using namespace std;
class Student{
   public:
      int rollno; //Attribute
      void print(){ //Function
         cout<<"RollNo:"<<rollno<<endl;
      }
      Student(){
         cout<<"Default constructor is called"<<endl;
      }
      Student(int rollno){
         cout<<"Parameterized constructor is called"<<endl;
         rollno=rollno;
      }
};
int main(){ //Start of the main function
   Student s1(50);
   s1.print();
}
```

→ Garbage Value

rollno = rollno;

Arg

Object of class

rollno = rollno;

Object's rollno = rollno;

Pointer : "this pointer"

this->rollno = rollno; ≡ s1.roll, s2.roll, s3.roll

Constructor is called with s1 : this → s1

s2: this → s2

```
#include<iostream> //Header File
using namespace std;
class Student{
   public:
      int rollno; //Attribute
      void print(){ //Function
         cout<<"RollNo:"<<rollno<<endl;
      }
      Student(){
         cout<<"Default constructor is called"<<endl;
      }
      Student(int rollno){
         cout<<"Parameterized constructor is called"<<endl;
         this->rollno=rollno;
      }
      int Func(){
         return this->rollno;
      }
};
int main(){ //Start of the main function
   Student s1(50);
   // s1.print();
   // Student s2(75);
   // s2.print();
   cout<<s1.Func();

}
```

s1.func() → s1.rollno

return this->rollno; — this: Calling object

s2.func()

s2. func( )

↓

s2.rollno